# Chapter 4

# Measurements for Software Aging

Roberto Pietrantuono*, Javier Alonso† and Kalyan Vaidyanathan‡

*Università degli Studi di Napoli Federico II, Italy*
*Amazon, Seattle, USA*
*BAE Systems, San Diego, USA*
**roberto.pietrantuono@unina.it*
*†jjava@amazon.com*
*‡kalyan.vaidyanathan@baesystems.com*

Considerable attention has been devoted to the analysis of software aging based on measurements from real systems. This approach aims to infer the presence of aging trends (aging detection) and to forecast the future evolution of such trends to determine the optimal rejuvenation time through an analysis of collected data.

This chapter will cover the main methods adopted for the analysis and detection of software aging phenomena based on measurements. It will discuss the methods for trend detection, estimation, forecasting, as well as data manipulation.

These methods can be classified as *threshold-based* approaches, *statistical* approaches for time series analysis and *machine learning* approaches for aging state classification and failure prediction. Mathematical details of the discussed techniques will be described in Appendix A-2.

## 4.1. Introduction

A significant amount of effort has been dedicated by academia as well as industry [1] to the analysis of software aging indicators (i.e., hardware and software metrics) primarily due to the fact that the software aging phenomenon can easily be observed in a complex enough system. These indicators are used to determine the existence of the phenomenon, and once this has been determined, the remaining time to life of the system under the effect of software aging is estimated, with the ultimate aim of determining the *optimal* moment to trigger the software rejuvenation process.

Depending on the techniques applied on the measurements collected from the system to estimate the presence of software aging and the estimation of the time to failure, we can broadly classify measurement-based software rejuvenation into three major approaches: the threshold-based approach, the statistics-based approach and the machine learning based approach.

In all of these approaches, a variety of system metrics such as memory usage, CPU usage, number of system requests, response time and file descriptors used are monitored. These metrics are used to determine which parts of the system are affected by the aging phenomena, to determine the time until the system reaches an unacceptable state, and finally to estimate the optimal moment to initiate the recovery process. The above-mentioned approaches differ in the way the system metrics are treated — thereby resulting in substantial differences in terms of accuracy and limitations, and in the way prior knowledge about the system is captured.

Threshold-based approaches are based solely on applying some level of prior knowledge of the system and the metric (or metrics) that exhibit the aging effect, and determining a threshold. If an aging metric reaches the pre-determined threshold value, the rejuvenation process is triggered. These techniques are usually based on some level of expert human knowledge and/or historical data.

The second group of techniques called the statistics-based approach use traditional statistical methods and tests. Statistical approaches use techniques like trend detection or time series analysis to determine if the aging phenomena actually exist and to estimate the time to failure of the system. This approach may be further categorized depending on the actual statistical techniques applied.

Finally, machine learning approaches leverage various machine learning algorithm families to try and predict system crashes due to software aging phenomena based on the indicators generated by the system such as memory usage, file descriptors and CPU usage.

This chapter devotes one section to each of the approaches discussed above. It can be argued that some statistical techniques can also be considered part of the machine learning family of approaches or that some approaches present hybrid solutions involving techniques from two or more groups of approaches.

This level of overlap can make it hard to classify each individual technique. However, in the opinion of the authors of this chapter, this proposed classification represents the best way to understand the idiosyncrasies and pros and cons of each family. So, when the reader confronts any new (hybrid or not) solution, they would be in a good spot to analyze it and understand the implications of such a solution, even if it involves techniques from more than one method presented here.

The rest of the chapter is thus organized as follows: Section 4.2 focuses on the threshold-based approaches. Section 4.3 analyzes the statistical approaches and describes the major techniques used within this field. Section 4.4 describes the main machine learning approaches used to predict when the system is most likely to crash due to the aging phenomena. Section 4.5 assists the reader by providing a comprehensive list of metrics that have been historically linked with the aging phenomena. The main goal of this section is to provide a good repository of well-known aging metrics. Section 4.6 concludes the chapter with some final thoughts.

## 4.2. Threshold-Based Approaches

Threshold-based approaches define thresholds for one or more aging indicators and rejuvenation is triggered when the monitored indicators exceed these thresholds. Aging indicators used here may refer to resource consumption levels. With this approach, it can sometimes be hard to assess how to trade-off between aging-related system crashes and the triggering of superfluous rejuvenation events.

An example of this approach is presented in the work by Silva *et al.* [2], which adopts thresholds on mean response time and on quality of service indicators. The authors propose a rejuvenation approach based on self-healing techniques that exploits virtualization to optimize recovery. They implemented a rejuvenation framework called *VM-Rejuv*, in which an *Aging Detector* module is designed to detect aging conditions based on the mentioned thresholds. Another work by Silva *et al.* [3] is a further example of the threshold-based approach; the paper presents an analysis of software aging in a SOAP-based server, in which a dependability benchmarking study is conducted to evaluate some SOAP-RPC implementations, focusing in particular on Apache Axis, where they detected the presence of aging by parameter monitoring.

Thresholds are also adopted by Matias *et al.* [4] to evaluate the aging effects in the Apache web server based on a controlled experiment. The memory consumed by Apache is observed together with three controllable workload parameters: page size, page type (dynamic or static) and request rate. The authors adopt thresholds on the usage of virtual memory as aging indications.

Threshold-based approaches can be combined with other approaches. For instance, a threshold could be established for some parameters for which a statistical or machine learning approach is difficult to apply (e.g., because of the characteristics of the observations such as high noise or variability), while time-series or machine learning approaches could be applied for other indicators. Such an approach has been applied on the Eucalyptus cloud computing platform: Araujo *et al.* [5] combined the threshold-based approach (with a threshold on memory utilization) with time-series analysis, implementing a rejuvenation policy in Eucalyptus by using multiple thresholds and forecasting with time series analysis models. The time-series models adopted in this work are the linear model, quadratic model, growth curve model, and S-curve trend model.

In related work, Avritzer *et al.* [6] applied the central limit theorem to customer response time samples to account for response time variability. Then, this was used to provide a mechanism to implement a trade-off between the detection of performance degradation transients and software rejuvenation events. The authors implement a series of varying buckets to store response time sample measurements that exceed the response time threshold. If a sample measurement was found to be less than the response time threshold, that would be used to cancel a previous sample that exceeded the response time threshold. The central limit theorem is used to show that even for a small number of samples, the response time distribution approaches a symmetric distribution over the mean response time [7]. If a given bucket sample storage has an overflow condition, the system declares the detection of performance degradation, creating a new bucket for sample storage, but not triggering a software rejuvenation event. Therefore, the system designers had the ability to postpone software rejuvenation, if the performance degradation was found to be related to a transient load increase. In contrast, if several buckets showed the overflow condition in sequence, the software aging detection mechanism would have an increased level of certainty that the detected performance degradation was not transient and that software rejuvenation would be beneficial.

## 4.3. Statistical Approaches

A widely used technique for the measurement-based approach is time series analysis. Variables potentially related to software aging, such as system resource consumption or user-perceived performance indicators, are monitored periodically over time, and the resulting time series data are analyzed to assess the presence of aging phenomena. A time series analysis for aging assessment addresses two main problems: 1) detecting the presence of a degradation trend, and, if such a degradation trend is present, 2) quantitatively estimating the extent of the degradation and its characteristics (e.g., seasonality in data). The problems are approached using *statistical trend detection* and *trend estimation* techniques, respectively. Trend detection applies statistical hypothesis testing to determine whether the values within a time series (i.e., a series of observations of a random variable) generally increase (or decrease) over a long period of time. The statistical test is used to make conclusive statements about the presence or absence of software aging trends with a desired statistical confidence. Trend estimation allows one to quantitatively characterize the long-term component of a time series.

The most used *trend detection* techniques in software aging research are, by far, the Mann-Kendall test and the seasonal Kendall test. They are hypothesis tests (for non-periodic and periodic time series, respectively) that are used to accept/reject the hypothesis of no trend in data. Being non-parametric tests, they do not make assumptions regarding the distribution of data and are more robust to outliers than parametric tests; on the other hand, they are generally conservative, in other words, some actual trend could be missed. The Mann-Kendall (and seasonal Kendall) test is usually applied in conjunction with Sen's (and the seasonal Sen's) trend estimation procedure for *trend estimation*. Sen's procedure is also a non-parametric linear regression technique (also insensitive to outliers) that fits a linear model and computes the rate at which the samples increase over time by simply estimating the slope as the median of all slopes between paired values [8, 9]. The combination of the Mann-Kendall test and Sen's estimator is a widely used approach and it has been reported in several papers from the reviewed aging research literature [10–13].

One of the first measurement-based analyses applying the Mann-Kendall test and Sen's procedure was by Garg *et al.* [10] Until then, observations of software aging had been mainly anecdotal. Instead of collecting metrics on failure events (which most research on software reliability relies on), the main idea behind this work was to periodically monitor and store data on the attributes of a software system that are indicative of the "health" of the system.

Using the SNMP (Simple Network Management Protocol) framework, a distributed resource monitoring tool was designed and developed. The goal was to monitor the health of Unix (Solaris) workstations at the OS level. The objects (or metrics) that were monitored fell broadly into seven categories — host details (that are constant), timestamps, OS resources, process states, file system resources, network resources and I/O resources. The SNMP agent ran on each monitored workstation while the monitoring station ran the SNMP manager and acted as a central data repository. Eight workstations were monitored for approximately two months with the frequency of data collection set to fifteen minutes.

Some of the questions that the authors attempted to answer were (a) is aging present (b) how do the metrics behave over time (c) can observed failures be related to specific metric values and (d) can software aging be quantified? Linear and periodic dependency analysis (autocorrelation and harmonic analysis) and trend detection and estimation techniques (the seasonal Kendall test) were used to answer these questions. Daily and weekly dependencies were observed in many of the monitored variables. Smoothing techniques (non-parametric local weighted regression) and trend detection (the seasonal Kendall test) were applied to the time-series data to detect global trends that pointed to evidence of software aging.

The major contributions of this paper were aging quantification and the estimations of time to failure due to aging. The quantification aspect was addressed by calculating a slope metric using Sen's non-parametric method for each metric and the corresponding time to failure estimation was performed using a simple linear projection, given the minimum/maximum of the metric of interest.

The above work was extended by Vaidyanathan and Trivedi [14], where a measurement-based model was proposed to estimate resource exhaustion times based on time as well as system workload state. The same setup as [10] was used to monitor and collect Unix data. Statistical clustering techniques were employed to identify different workload states and a semi-Markov state space model was developed. This approach can also be understood as a machine learning approach. For each of the monitored metrics of interest, a reward function was computed based on the exhaustion rate of these metrics (resources) at each of the identified workload states. The expected accumulated reward was used an indicator of the resource usage trend and estimates of failure times were computed using this. This work clearly demonstrated the relation between system workload and resource exhaustion and supports prior studies that showed that workload does affect system reliability/availability. Not only were the system workload dynamics captured in the model but also the effect of workload on resource usage was quantified by means of reward rates or slopes in the model. In doing this, the software aging phenomenon was validated with respect to resource exhaustion. This workload-based model was further expanded in [15] by feeding the results to a higher level availability model that accounts for failure followed by reactive as well as proactive recovery. Optimal rejuvenation schedules are derived that minimize downtime cost and maximize availability.

Grottke *et al.* [11] have analyzed the performance degradation in the Apache Web Server by sampling the web server's response time to predefined HTTP requests at fixed intervals, using a similar procedure as the one adopted by Garg *et al.* They have also analyzed the seasonal patterns, where trend analysis accounts for the possible presence of seasonal variation in data.

More recently, Zheng *et al.* [16] proposed an alternative to the conventional (seasonal) Mann-Kendall test and Sen's procedure. The authors point out a few areas where improvements can be made to the Mann-Kendall test and Sen's trend estimator (e.g., suitability for linear trends only, sensitivity to noise and computational complexity) and propose to use a modified version of the Cox-Stuart test for trend detection and the iterative Hodrick-Prescott Filter for (linear and non-linear) trend estimation. The Cox-Stuart test is based on the sign test between pairs of a series of observations (a modified version is proposed for periodic time series). The Hodrick-Prescott filter does not presume the structure of the trend (e.g., linear trend); it solves a "penalized" spline model, fitting the raw time series to be a more smoothened representation, by using a smooth parameter that penalizes the variability of the trend component, thus controlling the trade-off between the goodness of fit and smoothness (the larger its value, the smoother the trend component). This test is also insensitive to periodicity.

Together with the conventional Mann-Kendall test for trend detection, Li *et al.* [17] used time-series ARMA/ARX models for trend estimation on the Apache Web Server to estimate the resource exhaustion. Compared with the linear regression and extended linear regression models, the ARX model incurs higher initial overhead, but once the model parameters are established, these parameters can be used for prediction without the need for frequent model re-calibration.

ARIMA (Autoregressive Integrated Moving Average) and Holt-Winters (Triple Exponential Smoothing) models have been used by Magalhaes and Silva [18], wherein the authors develop a framework for detection of performance anomalies caused by aging, which is targeted to web and component-based applications. In particular, the framework monitors application/system parameters, used to determine the correlation between the application response time and the input workload, in turn used to train machine learning (ML) algorithms. At run-time, parameters collected by monitoring are estimated by ARIMA and Holt-Winters algorithms, and the estimations are classified by the trained ML algorithms to determine if the application may experience performance anomalies.

Time-series analysis has also been adopted to study the relationship between software aging and workload in complex systems, including the Linux Kernel code [19] and the Java Virtual Machine [20]. In both cases, the analysis of workload parameters is used to provide indications of potential sources of software aging, by highlighting the subsystems whose parameters are correlated to the experienced aging trends. In the latter, parametric trend detection was used for aging detection, specifically the conventional *t-student* test. Then, Principal Component Analysis (PCA) followed by multiple linear regression was also used to remove first-order correlation among predictors and then to provide linear estimates of aging trends, so as to reduce the problem of multicollinearity. Clustering is also preliminarily exploited to separate out different workload states possibly exhibiting different trends.

In the work by Bovenzi *et al.* [21] the authors adopt a Design of Experiment (DoE) technique that is used to assess the presence of aging under different workload configurations in several applications, such as the James mail server, a middleware for an air traffic control system, the Apache web server, and the MySQL DBMS. A series of controlled long-running experiments are planned and executed, and the resulting time series are analyzed by the conventional Mann-Kendall test and Sen's procedure followed by the Analysis of Variance (ANOVA) to assess the impact of various workload factors on software aging. The impact of concurrency bugs on aging is also assessed using this technique.

Non-linear time-series analyses are used by Araujo *et al.* [5] Four different time-series models have been used to schedule software rejuvenation: 1) the linear model, 2) the quadratic model, 3) the exponential growth model, and, 4) the model of the Pearl-Reed logistic. They have been adopted to predict memory consumption trends on the Eucalyptus cloud computing framework. One further paper considering nonlinear models is by Jia *et al.* [22] in which aging is studied in the Apache server by constructing a dynamic model to describe the software aging process following the method of nonlinear dynamic inversion. The software aging process is shown to be nonlinear and chaotic.

In the best practice guide by Hoffmann *et al.* [23], multivariate nonlinear models (support vector machines, radials, and universal basis functions) have been compared with multivariate linear models. The former ones have shown better performance than linear models in the benchmarking case studies.

Finally, Chen *et al.* [24] propose a new aging metric based on Multidimensional Multi-scale Entropy (MMSE) for trend analysis and aging-related failure prediction. PCA is leveraged to select variables from an initial set of multiple aging indicators. Using a normalized time series of the selected indicators, the MMSE metric is computed, where a greater entropy is demonstrated to relate to a more severe aging phenomenon.

## 4.4. Machine Learning Approaches

As stated in the previous section, there are several works that use time series approaches as a first step in the area of machine learning. If we consider software aging phenomena as a result of two factors (the system resource presenting the aging bug, and time), time series approaches could be considered ideal. However, these approaches are drastically limited by the need to have previous knowledge about the resource that is affected by the aging phenomena and the affected performance metric (e.g., response time, throughput). This represents an increasingly higher barrier to adopting time series approaches with the increasing complexity of the systems and the interdependency between resources at different system layers.

In this scenario machine learning approaches come into play. Machine learning approaches are a more sophisticated form of data analysis, which adopts machine learning algorithms (e.g., classifiers and regressors) to identify trends (e.g., estimating time to failure) and to classify a system state as robust or failure-prone.

Pattern recognition methods have been used by Cassidy *et al.* [25] to predict software aging in shared memory pool latch contention in large OLTP servers. The approach applies non-linear, non-parametric regression to a large set of system variables, and analyzes the residual error between the predicted and the actual system values using a sequential probability ratio test, in order to predict the onset of software aging effects. The obtained results have shown that these methods could be used to detect significant deviations from "standard" behavior with a 2-hour early warning.

We subdivide the machine learning approaches into two major families based on the type of machine learning algorithm used: regression approaches and classification approaches.

The former approach is focused on trying to predict the time to exhaustion (TTE) of the system resource or resources due to aging phenomena. The later approach is focused on determining the state of the system (i.e., stable vs non-stable state).

There are several published papers within the domain of predicting the time to crash due to software aging. For example, Alonso *et al.* [26] compare different regression algorithm families (i.e., regression trees, linear regression and hybrids). The authors focus on comparing these algorithms within the different scenarios and multiple aging phenomena involved. The conclusion arrived at by the authors was that the aging phenomena was better modeled by a hybrid model (i.e., MP5) between Decision Trees and Linear Regression, since the aging phenomena caused by resource exhaustion due to bugs in the software (i.e., memory leaks or threads unreleased after use) can be modeled by linear piecewise models capturing different aging slopes or speeds. In addition to the above, three different machine learning algorithms (a naive Bayes classifier, decision trees and a neural network model) have been also used in combination with time-series models to predict aging in web applications [18].

They automatically built the models relating several system variables (e.g., the number of connections and throughput) to aging trends, based on the observation that software aging trends can be approximated using a piecewise linear model. The models were trained using data samples collected in preliminary experiments, and were used to predict the TTE of system resources in conditions different than the ones observed during the training phase.

An example of the application of machine learning classifier algorithms is presented by Alonso *et al.* [27]. The paper compares a large set of families (i.e., random forest, decision trees, LDA/QDA, Naive Bayes, Support Vector Machines and K-nearest neighbors) to predict state in the context of a three-tier J2EE system.

Andrzejak and Silva [28] compare four classification algorithms: J48 (a decision tree based algorithm), Navie Bayes, Support Vector Machines (i.e., SMO) and ZeroR (The 0/R classifier). The authors compare these four algorithms in software aging scenarios caused by only one metric with a constant software aging injection rate. The experimental evaluation suggested that all these four main families of classification techniques perform similarly, even though SMO presents better overall results.

An approach not exactly falling within this area but still related to it is to use reinforcement learning for aging estimation. Eto, Dohi and Ma [29] used reinforcement learning to estimate the optimal rejuvenation schedule adaptively, i.e., by considering runtime data to update the estimation. Thus, their estimation technique does not require a complete knowledge of system failure (degradation) time distribution in the operational phase, even if the underlying software state transition is governed by models, i.e., by CTMCs or SMPs. This is an example of a *hybrid* approach, wherein a measurements-based approach is used in conjunction with a model-based one [30].

## 4.5. Relevant Software Aging Metrics: Beyond Memory Leaks

Aging indicators are an important area of study for measurement-based approaches. The correct identification of metrics representing the aging of the system is of paramount importance in order to have a clear assessment of the system health's state. Aging indicators can refer to both *resource usage* and to *performance*. The following classes of aging indicators are commonly used in measurement-based aging analysis techniques:

- **Memory consumption**: metrics related to memory consumption are by far the most commonly used metrics for monitoring resource depletion. Empirical evidence showed that free memory exhibits the shortest Time to Exhaustion (TTE) among system resources [10] and that memory management defects are a significant cause of aging failures [31] and of software failures in general [32]. Therefore, several measurement-based studies analyze aging phenomena affecting free memory by measuring the amount of *free physical memory* and *swap space* [11, 14, 33], and applying time series and statistical models to these variables.
- **Performance degradation**: Measurement-based techniques often refer to *performance degradation* in software systems affected by aging, which is the other symptom (aside from resource depletion) caused by aging. Performance degradation is indeed related to memory usage: for instance, the consumption of physical memory increases the time required by memory allocation procedures and garbage collection mechanisms, since their computational complexity is a function of the number of memory areas that have been allocated [20, 34, 35]. But performance degradation can be also due to other, more complex and hard-to-reproduce, aging causes such as fragmentation or concurrency management [21, 36]. Common indicators are response time, latency, throughput, transaction rate or, less commonly, the number of SLA (Service Level Agreement) violations. In the presence of this kind of phenomena, software rejuvenation can be triggered when the quality of service (e.g., in terms of response time or throughput) is below a given threshold.

- **Other resource consumption**: Software aging can impact several other resources types. Besides memory-related resources (e.g., physical memory, virtual memory, swap space, cache memory), the reviewed literature also addressed the following other types of resources:
  - — filesystem-related resources, such as stream descriptors and file handles [10, 37, 38]
  - — storage, whose space may be consumed by bad management [39]
  - — network-related resources, such as socket descriptors [37]
  - — concurrency-related resources, such as locks, threads and processes [10, 21, 38]
  - — application-specific resources, such as DBMS shared pool latches [25] and OSGi references [40].

  In many cases, the approach is not constrained to a specific resource, but is focused on detecting incorrect API usage and incorrect exception handlers that may cause a resource leakage. For instance, in the work by Zhang *et al.*, the authors present an approach that dynamically mines resource usage patterns by monitoring API calls, and provides an experimental evaluation of open source programs based on the Java I/O and concurrency APIs [38]. A frequent kind of resource leak in Java programs is represented by sockets and file handles, due to faulty exception handlers that do not release these resources [37, 38]. Other resources can also be affected by software aging depending on the kind of system, such as free disk space in DBMS [39]. In some cases, a wider set of resources can be analyzed. The above-mentioned example by Garg *et al.* applies time series analysis on a network of UNIX workstations monitored on several resources (related to virtual memory, the OS kernel, the file system, the disk, and the network), noticing a statistically significant trend in the process table size and in the file table size (although their time-to-exhaustion was lower than that of free memory). Other resources include CPU utilization, power consumption, and the number of threads/processes. In cloud systems, indicators can be measured at any layer of the virtualization technology stack, e.g., at the application or the OS level within the VM layer, at the VM layer to probe the state and the resource consumption of the VMs (e.g., for load balancing, scaling, VM migration and rejuvenation decisions), as well as at the VMM layer. Besides system resources, other indicators of interest are related to VMs, for instance, the number of VM new allocations/releases, the time to start/stop VMs, or to the time to migration.

A recent field in which software rejuvenation is being studied is related to **security attacks** [41, 42], that is, attempts by malicious users to access unauthorized resources, to cause their gradual leakage or to make the system unavailable. In fact, security attacks may take place and gradually compromise a system over a long period of time (e.g., password theft through brute force guessing, or flood attacks that trigger software aging phenomena), which can be mitigated by periodically rejuvenating a system, such as by changing cryptographic keys, by restarting compromised processes, and by randomizing the location of data and instructions in memory [43–48]. Currently, the aging rate is assumed at design time [43, 49] or is based on imperfect attack/intrusion detectors that could raise false alarms and miss attacks [50, 51].

Other aging indicators may be aimed at capturing effects like the *accumulation of numerical errors* [52] and *memory fragmentation* [52, 53]. These kinds of aging effects are not necessarily caused by bugs in the software, but are related to the nature of floating-point arithmetic and memory allocation algorithms respectively.

## 4.6. Conclusions

This chapter described the main approaches to dealing with software aging based on *measurements*. Unlike model-based approaches, measurements-based techniques rely on observations from real systems, with the goal of *detecting* whether the system is in a failure-prone state due to software aging, *forecasting* the time-to-aging-failure, and *planning* software rejuvenation accordingly.

The advantage of such solutions lies in the possibility of gathering accurate and detailed information about the aging state of the system under analysis, and the use of real data to estimate metrics and model parameters.

However, while the techniques described in this chapter can be applied to any system for which monitoring is possible, it is also true that the gathered observations from one system are difficult to generalize. A good example of this difficulty is that of trained machine learning models, which may work accurately only for the system that was used to generate the training data.

Moreover, at design time there is no data available to determine the potential sources of possible aging behaviors — a real system is needed in order for data to be collected. This requires a tuning of the technique used based on the system to be analyzed, which considers the indicators that are more likely related to the aging phenomena of that specific system — hence more explanatory of aging behaviors — and that can be monitored with acceptable overhead.

Due to the underlying limitations of the measurements-based approaches, it is natural to combine measurements-based approaches with model-based approaches. Real data can be used to estimate the parameters for the models. So, combining both approaches increases our capabilities to study the effect of the software aging on a given system in much more detail than is possible with simple observations.

In the future, we expect the impact of the measurement-based aging analysis technique to increase in the SAR literature. We expect to see an increased use of machine learning techniques because they represent a hybrid between model-based and measurements-based approaches.

Indeed, as systems become more and more complex, several indicators will need to be analyzed to detect software aging. Moreover, it will become increasingly difficult to distinguish between software aging and the expected fluctuations of the system performance, because of inter-relations among multiple parameters [6, 7].

All these call for more complex statistical techniques able to analyze the cause-effect relations (and not only correlations) among multiple indicators and to spot those critical patterns relevant for aging identification in complex systems. This is a challenge of great importance due to the increasing impact of aging problems in complex systems.

## References


1. J. Alonso, A. Bovenzi, J. Li, Y. Wang, S. Russo, and K. Trivedi. (2013). Software rejuvenation: Do IT and telco industries use it? In *Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW), Dallas, Texas, USA, November 27–30*, pp. 299–304. doi: 10.1109/ISSREW.2012.96.
2. L. Silva, J. Alonso, and J. Torres. (2009). Using virtualization to improve software rejuvenation. *IEEE Transactions on Computers*, **58**(11), 1525–1538.
3. L. Silva, H. Madeira, and J. Silva. (2006). Software aging and rejuvenation in a SOAP-based server. In *Proceedings of the 5th IEEE International Symposium on Network Computing and Applications (NCA 2006), Cambridge, Massachusetts, USA, July 24–26*, pp. 56–65.
4. R. Matias Jr. and P. Filho. (2006). An experimental study on software aging and rejuvenation in web servers. In *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Chicago, Illinois, USA, September 17–21*, pp. 189–196.
5. J. Araujo, R. Matos, P. Maciel, F. Vieira, R. Matias, and K. Trivedi. (2011). Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. In *Third International Workshop on Software Aging and Rejuvenation (WoSAR), Hiroshima, Japan, November 29–December 2*, pp. 38–43.

6. A. Avritzer, A. Bondi, and E. J. Weyuker. Ensuring stable performance for systems that degrade. In *Proceedings of the 5th International Workshop on Software and Performance (WOSP '05), Palma, Illes Balears, Spain, July 12–14*, pp. 43–51. ISBN 1-59593-087-6. doi: 10.1145/1071021.1071026.
7. A. Avritzer, A. Bondi, M. Grottke, K. S. Trivedi, and E. J. Weyuker. (2006). Performance assurance via software rejuvenation: Monitoring, statistics and algorithms. In *International Conference on Dependable Systems and Networks (DSN'06), Philadelphia, Pennsylvania, USA, June 25–28*, pp. 435–444. doi: 10.1109/DSN.2006.58.
8. P. K. Sen. (1968). Estimates of the regression coefficient based on Kendall's Tau. *Journal of the American Statistical Association*, **63**(324), 1379–1389. ISSN 01621459.
9. H. Theil. (1992). A rank invariant method of linear and polynomial regression analysis. In Raj, B. and Koerts, J. (eds.). *Henri Theil's Contributions to Economics and Econometrics: Econometric Theory and Methodology*. Dordrecht, Netherlands: Springer. ISBN 978-94-011-2546-8. doi: 10.1007/978-94-011-2546-8_20.
10. S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi. (1988). A methodology for detection and estimation of software aging. In *Proceedings of the Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, November 4–7*, pp. 283–292. doi: 10.1109/ISSRE.1998.730892.
11. M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi. (2006). Analysis of software aging in a web server. *IEEE Transactions on Reliability*, **55**(3), 411–420. ISSN 0018-9529. doi: 10.1109/TR.2006.879609.
12. A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo. (2011). Workload characterization for software aging analysis. In *Proceedings of the 22nd International Symposium on Software Reliability Engineering (ISSRE), Tokyo, Japan, November 14–17*, pp. 240–249.
13. K. S. Trivedi, K. Vaidyanathan, and K. Goseva-Popstojanova. (2000). Modeling and analysis of software aging and rejuvenation. In *Proceedings of the 33rd Annual Simulation Symposium (SS 2000), Washington, DC, USA, April 16–20*, pp. 270–279.
14. K. Vaidyanathan and K. Trivedi. (1999). A measurement-based model for estimation of resource exhaustion in operational software systems. In *Proceedings of the 10th International Symposium on Software Reliability Engineering, Boca Raton, Florida, USA, November 1–4*, pp. 84–93. doi: 10.1109/ISSRE.1999.809313.
15. K. Vaidyanathan and K. Trivedi. (2005). A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, **2**(2), 124–137.
16. P. Zheng, Y. Qi, Y. Zhou, P. Chen, J. Zhan, and M. R. T. Lyu. (2014). An automatic framework for detecting and characterizing performance degradation of software systems. *IEEE Transactions on Reliability*, **63**(4), 927–943. ISSN 0018-9529. doi: 10.1109/TR.2014.2338255.
17. L. Li, K. Vaidyanathan, and K. Trivedi. (2002). An approach for estimation of software aging in a web server. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering, Nara, Japan, October 3–4*, pp. 91–102. doi: 10.1109/ISESE.2002.1166929.
18. J. Magalhaes and L. Silva. (2010). Prediction of performance anomalies in web-applications based-on software aging scenarios. In *Proceedings of the Second International Workshop on Software Aging and Rejuvenation (WoSAR), San Jose, California, USA, November 2*, pp. 1–7. doi: 10.1109/WOSAR.2010.5722095.
19. D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo. (2010). Software aging analysis of the Linux operating system. In *Proceedings of the 21st International*

*Symposium on Software Reliability Engineering (ISSRE) San Jose, California, USA, November 1–4*, pp. 71–80. doi: 10.1109/ISSRE.2010.24.
20. D. Cotroneo, S. Orlando, R. Pietrantuono, and S. Russo. (2013). A measurement-based ageing analysis of the JVM. *Software Testing, Verification and Reliability*, **23**(3), 199–239.
21. A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo. (2012). On the aging effects due to concurrency bugs: A case study on MySQL. In *Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering, Dallas, Texas, USA, November 27–30*, pp. 211–220. doi: 10.1109/ISSRE.2012.50.
22. Y.-F. Jia, L. Zhao, and K.-Y. Cai. (2008). A nonlinear approach to modeling of software aging in a web server. In *Proceedings of the 15th Asia-Pacific Software Engineering Conference, Beijing, China, December 3–5*, pp. 77–84. doi: 10.1109/APSEC.2008.38.
23. G. Hoffmann, K. Trivedi, and M. Malek. (2007). A best practice guide to resource forecasting for computing systems. *IEEE Transactions on Reliability*, **56**(4), 615–628. ISSN 0018-9529. doi: 10.1109/TR.2007.909764.
24. P. Chen, Y. Qi, X. Li, D. Hou, and M. Lyu. (2017). Arf-predictor: Effective prediction of aging-related failure using entropy. *IEEE Transactions on Dependable and Secure Computing*, **15**(4), 675–693. ISSN 1545-5971. doi: 10.1109/TDSC.2016.2604381.
25. K. Cassidy, K. Gross, and A. Malekpour. (2002). Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers. In *Proceedings of the International Conference of Dependable Systems and Networks, Bethesda, Maryland, USA, June 23–26*, pp. 478–482. doi: 10.1109/DSN.2002.1028933.
26. J. Alonso, J. Torres, J. Berral, and R. Gavalda. (2010). Adaptive on-line software aging prediction based on machine learning. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN), Chicago, Illinois, USA, June 28–July 1*, pp. 507–516. doi: 10.1109/DSN.2010.5544275.
27. J. Alonso, L. Belanche, and D. Avresky. (2011). Predicting software anomalies using machine learning techniques. In *Proceedings of the 10th IEEE International Symposium on Network Computing and Applications (NCA), Cambridge, Massachusetts, USA, August 25–27*, pp. 163–170.
28. A. Andrzejak and L. Silva. (2008). Using machine learning for non-intrusive modeling and prediction of software aging. In *Proceedings of the IEEE Network Operations and Management Symposium, Salvador, Bahia, Brazil, April 7–11*, pp. 25–32.
29. H. Eto, T. Dohi, and J. Ma. (2008). Simulation-based optimization approach for software cost model with rejuvenation, In Rong C., Jaatun M. G., Sandnes F. E., Yang L. T., Ma J. (eds.). *Autonomic and Trusted Computing.* ATC 2008. Lecture Notes in Computer Science, vol 5060. Berlin/Heidelberg, Germany: Springer.
30. D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo. (2014). A survey of software aging and rejuvenation studies. *Journal of Emerging Technologies in Computing Systems*, **10**(1), 8:1–8:34. ISSN 1550-4832. doi: 10.1145/2539117.
31. D. Cotroneo, R. Natella, and R. Pietrantuono. (2012). Predicting aging-related bugs using software complexity metrics. *Performance Evaluation*, **70**(3), 163–178.
32. M. Sullivan and R. Chillarege. (1991). Software defects and their impact on system availability — a study of field failures in operating systems. In *Proceedings of the Twenty-First International Symposium on Fault-Tolerant Computing (FTCS-21), Montreal, Quebec, Canada, June 25–27*, pp. 2–9.
33. M. Ficco, R. Pietrantuono, and S. Russo. (2017). Aging-related performance anomalies in the Apache Storm stream processing system. *Future Generation Computer Systems*, **86**, 975–994.

34. G. Carrozza, D. Cotroneo, R. Natella, A. Pecchia, and S. Russo. (2010). Memory leak analysis of mission-critical middleware. *Journal of Systems and Software*, **83**(9), 1556–1567.
35. T. Ferreira, R. Matias, A. Macedo, and L. Araujo. (2011). An experimental study on memory allocators in multicore and multithreaded applications. In *Proceedings of the 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Gwangju, Korea, October 20–22*, pp. 92–98. doi: 10.1109/PDCAT.2011.18.
36. D. G. Cavezza, R. Pietrantuono, J. Alonso, S. Russo, and K. S. Trivedi. (2014). Reproducibility of environment-dependent software failures: An experience report. In *Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering, Naples, Italy, November 3–6*, pp. 267–276. doi: 10.1109/ISSRE.2014.19.
37. W. Weimer. (2006). Exception-handling bugs in Java and a language extension to avoid them. In Dony C., Knudsen J. L., Romanovsky A., Tripathi A. (eds.) *Advanced Topics in Exception Handling Techniques*. Lecture Notes in Computer Science, vol. 4119. Berlin/Heidelberg, Germany: Springer.
38. H. Zhang, G. Wu, K. Chow, Z. Yu, and X. Xing. (2011). Detecting resource leaks through dynamical mining of resource usage patterns. In *Proceedings of the 41st IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W) Hong Kong, China, June 27–30*, pp. 265–270.
39. A. Bobbio, M. Sereno, and C. Anglano. (2001). Fine grained software degradation models for optimal rejuvenation policies. *Performance Evaluation*, **46**(1), 45–62.
40. K. Gama and D. Donsez. (2008). Service coroner: A diagnostic tool for locating OSGi stale references. In *Proceedings of the 34th Euromicro Conference on Software Engineering and Advanced Applications, Parma, Italy, September 3–5*, pp. 108–115.
41. A. Avritzer, R. G. Cole, and E. J. Weyuker. (2010). Methods and opportunities for rejuvenation in aging distributed software systems. *Journal of Systems and Software*, **83**(9), 1568–1578. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2010.05.026.
42. A. Avritzer, R. G. Cole, and E. J. Weyuker. (2007). Using performance signatures and software rejuvenation for worm mitigation in tactical manets. In *Proceedings of the 6th International Workshop on Software and Performance (WOSP '07), Buenos Aires, Argentina, February 5–8*, pp. 172–180. ISBN 1-59593-297-6. doi: 10.1145/1216993.1217023.
43. P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Verissimo. (2010). Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel and Distributed Systems*, **21**(4), 452–465. ISSN 1045-9219. doi: 10.1109/TPDS.2009.83.
44. A. Tai, K. Tso, W. Sanders, and S. Chau. (2005). A performability-oriented software rejuvenation framework for distributed applications. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks, Yokohoma, Japan, June 28–July 1*, pp. 570–579. doi: 10.1109/DSN.2005.12.
45. A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T. Uribe. (2003). An architecture for an adaptive intrusion-tolerant server. In Christianson B., Crispo B., Malcolm J. A., and Roe M. (eds.). *Security Protocols*. Security Protocols 2002. Lecture Notes in Computer Science, vol 2845. Berlin/Heidelberg, Germany: Springer.
46. B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. (2006). N-variant systems: A secretless framework for security through diversity. In *Proceedings of the 15th Conference on USENIX Security*

*Symposium, Vancouver, British Columbia, Canada, July 31–August 4, Volume 15*, p. 9. USENIX Association, (2006).

47. Y. Huang, D. Arsenault, and A. Sood. (2006). SCIT-dNS: Critical infrastructure protection through secure DNS server dynamic updates. *Journal of High Speed Networks*, **15**(1), 5–19.
48. T. Roeder and F. Schneider. (2010). Proactive obfuscation. *ACM Transactions on Computer Systems (TOCS)*, **28**(2), 4.
49. Q. Nguyen and A. Sood. (2009). Quantitative approach to tuning of a time-based intrusion-tolerant system architecture. In *Proceedings of the 3rd Workshop on Recent Advances in Intrusion-Tolerant Systems, Lisbon, Portugal, June 29–July 2*, pp. 132–139.
50. K. Aung, K. Park, and J. Park. (2005). A model of ITS using cold standby cluster. In Fox E. A., Neuhold E. J., Premsmit P., and Wuwongse V. (eds.). *Digital Libraries: Implementing Strategies and Sharing Experiences*. ICADL 2005. Lecture Notes in Computer Science, vol 3815. Berlin/Heidelberg, Germany: Springer.
51. A. Nagarajan and A. Sood. (2010). SCIT and IDS architectures for reduced data exfiltration. In *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W), Chicago, Illinois, USA, June 28–July 1*, pp. 164–169.
52. M. Grottke, R. Matias, and K. Trivedi. (2008). The fundamentals of software aging. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops, Seattle, Washington, USA, November 11–14*, pp. 1–6.
53. A. Macedo, T. Ferreira, and R. Matias. (2010). The mechanics of memory-related software aging. In *Proceedings of the 2010 IEEE Second International Workshop on Software Aging and Rejuvenation (WoSAR), San Jose, California, USA, November 2*, pp. 1–5. doi: 10.1109/WOSAR.2010.5722097.